1/5/2001, Version 0.6

Software Note

Booting User Software From Flash Chips

Preface

Often the HW/SW developer needs to design a solution that uses a single flash device as the source of the BIOS, OS and/or application. This document focuses on booting Linux from a single FLASH chip on the MachZ Integrated Development System (IDS). The theory provided in this document can also be applied for booting other operating systems such as Wind River s VxWorks or user-written special application programs.

The general approach given here is BIOS-independent, although the example utilizes MachZ specific features built into the latest MachZ Phoenix BIOS.

Introduction to Extension ROMs

Our flash-software booting approach relies on the Extension ROM scan system, which is a feature found in all AT-compatible PC-s.

A VIDEO BIOS on a common ISA video card can be considered an Extension ROM, although it is a special case that gets executed in the very early stage of the BIOS POST sequence. Other examples are: ROM-BASIC used on early IBM AT-s and all firmware on PCI or ISA extension boards (network interface controllers, SCSI controllers etc.).

During the Power-On-and-Self-Test (POST) sequence the BIOS performs a so-called ROM-scan sequence. The BIOS looks at the beginning of every 2kbyte block in the address region C8000h-F0000h to find a 55AA signature. When the signature is found, the next byte determines the size of the option ROM in 512-byte increments. For detecting corruption, the sum of all option-ROM bytes must be 100h. This is usually achieved by setting the last byte to 100h-(sum of all other option-ROM bytes). When the BIOS has validated the correctness of the extension ROM, it calls the routine, which starts at offset 03 of the option-ROM.

Table of Contents

P/N 9100-0067-00

1052 Elwell Court Palo Alto, CA 94303 • Tel: 800.683.5943 • www.zflinux.com

The routine contained in the option-ROM can do any imaginable task, but usually it just initializes the hardware to some known state and hooks interrupt vectors that will be used later by other operating-system services or user programs.

We are going to use this option-ROM code for booting Linux from FLASH. From now on, the code executed from our option-ROM will be called the Linux Loader (LL).

This process works for all possible HW configurations where the option-ROM can be resident in the same device as the BIOS, or in different chips on the motherboard. The only requirement is that the chip region where the option-ROM resides can be addressed during the POST sequence. This requires setting/routing the Chip-Select signals from the MachZ and correctly initializing the custom BIOS settings.

How the Linux Loader works

The Linux Loader (LL) copies the Linux Kernel from a Flash address to a RAM address that matches the address used when the Linux is started from a Hard Disk.

Also, depending on compiled-in options, the Kernel will know whether to mount its root file system from a RAM disk or from a Hard Disk partition. The RAM disk root file system is initially stored in flash as a compressed INITRD image and will be copied to the end of available RAM by the Linux Loader. The INITRD image in flash is preceded with a 4-byte length value, which indicates to the LL how many bytes to copy from Flash to RAM.

The internal workings of the Linux Loader are shown below. These same concepts can be used for launching other operating systems:

1. Linux Loader is invoked by a special extension-ROM scan routine that gains control just before the normal boot process and scans the memory-window defined in Phoenix BIOS Setup Utility ->Advanced -> Advanced Chipset Control -> ISA Memory Chip Select Setup mem_cs0 settings. The user can use these memory-window settings to map certain regions from flash memory to a desired address below the 1Mb boundary (by default part of the BIOS itself is mapped from the end of the 2Mb flash device to address E0000h using this chip-select signal). The mem_cs0 can be set to any needed value, because at the time-point when those settings are really needed, the BIOS is already shadowed and is not executed from the flash device any more.
2. As a first step after execution, the Linux Loader relocates itself to main working memory address 9B00:0000. This is done because in later processing steps the mem_cs0 memory window will be re-defined and if the LL code is still working at this point from the flash device (mapped to memory window below 1Mb), during the mem_cs0 redefinition it disappears from initial memory-window and a total system crash occurs.
3. LL initializes the serial port to allow for diagnostic messages.
4. Then LL will wait 3 seconds for user input. If the ESC key is pressed, the loader quits and the BIOS special extension-ROM scan routine gets control again. If the extension-ROM scan routine cannot find any other valid extension-ROMs in the defined search area (defined by mem_cs0 settings), the normal disk boot will occur.

5. After the 3 seconds have elapsed without the ESC key being pressed, the loading sequence starts. First the original mem_cs0 flash window settings are saved and this flash window is enlarged to 16MB using ZF Logic s chip-select programming features. This flash window will be visible for all available memory addresses not claimed by the memory controller (addresses reserved for RAM) or PCI devices. This happens because the ISA bus has a lower priority than all the other chip devices (memory controller or PCI controller). Therefore, at the end of the DRAM memory map, the mapped flash contents will be repeated over the entire upper memory space on every 16Mbyte boundary. For instance, the entire 16Mbyte Flash contents is visible in the address range 10000000h — 10FFFFFFh (and also at 11000000h-11FFFFFFh, 12000000h-12FFFFFFh etc.).

6. **Note:** To get access to the entire upper memory space, enable the A20 line.

7. Before the LL can copy large amounts of data from the extended memory to the lower RAM, it has to account for protected-mode operation. First it initializes the Global Descriptor Table (GDT) so that the data segment size will increase to 4GB instead of the normal 64KB. Then the LL will load the GDT, switch the processor to protected mode, set data segment DS and extra segment ES selectors with the previously defined 4GB data-range GDT entries, and switch the processor back to real mode again. This allows us to access the entire 4GB memory space in real mode. This will be true as long as the DS and ES registers are not overwritten.

8. Once the LL has access to the full memory space, it checks the Linux kernel s signature at Flash offset 202h (for our example, visible at memory address 10000202h). If the signature is not found, the LL increments the search address with 16Mb and checks again for a signature. If after 10 checks the kernel signature is not found (the signature is searched from addresses 10000202h,11000202h, 12000202h etc.), the LL restores the original memory window settings, prints out the Linux kernel setup signature not found message and returns.

9. At the start of the kernel image there is a bootstrap code and a kernel setup code. The setup code size is read from the kernel header data structures and all the code is copied from the previously found address (10000000h etc.) to 90000h in low memory.

10. The kernel loading address is read from the kernel headers and the kernel itself (which starts right after the setup code in Flash) and is then copied to the correct loading address in RAM. In the case of a normal kernel it will be loaded at address 10000h; in the case of a big kernel (made with make bzImage) it will be loaded at high memory address 100000h.

11. In order to load the Initrd image to RAM, the LL requests the detected top of memory system address from the South Bridge.

12. The Linux Loader requests that the Initrd image start at flash offset 80000h. Before the Initrd gets copied, the LL checks for its presence by reading the Initrd size value from flash offset 80000h (memory address 10080000h, 11080000h etc.). If the size value is 0 or 0FFFFFFFFh then the Initrd copying is skipped, otherwise the Initrd image is copied to the end of the detected RAM without the 4-byte length indicator.

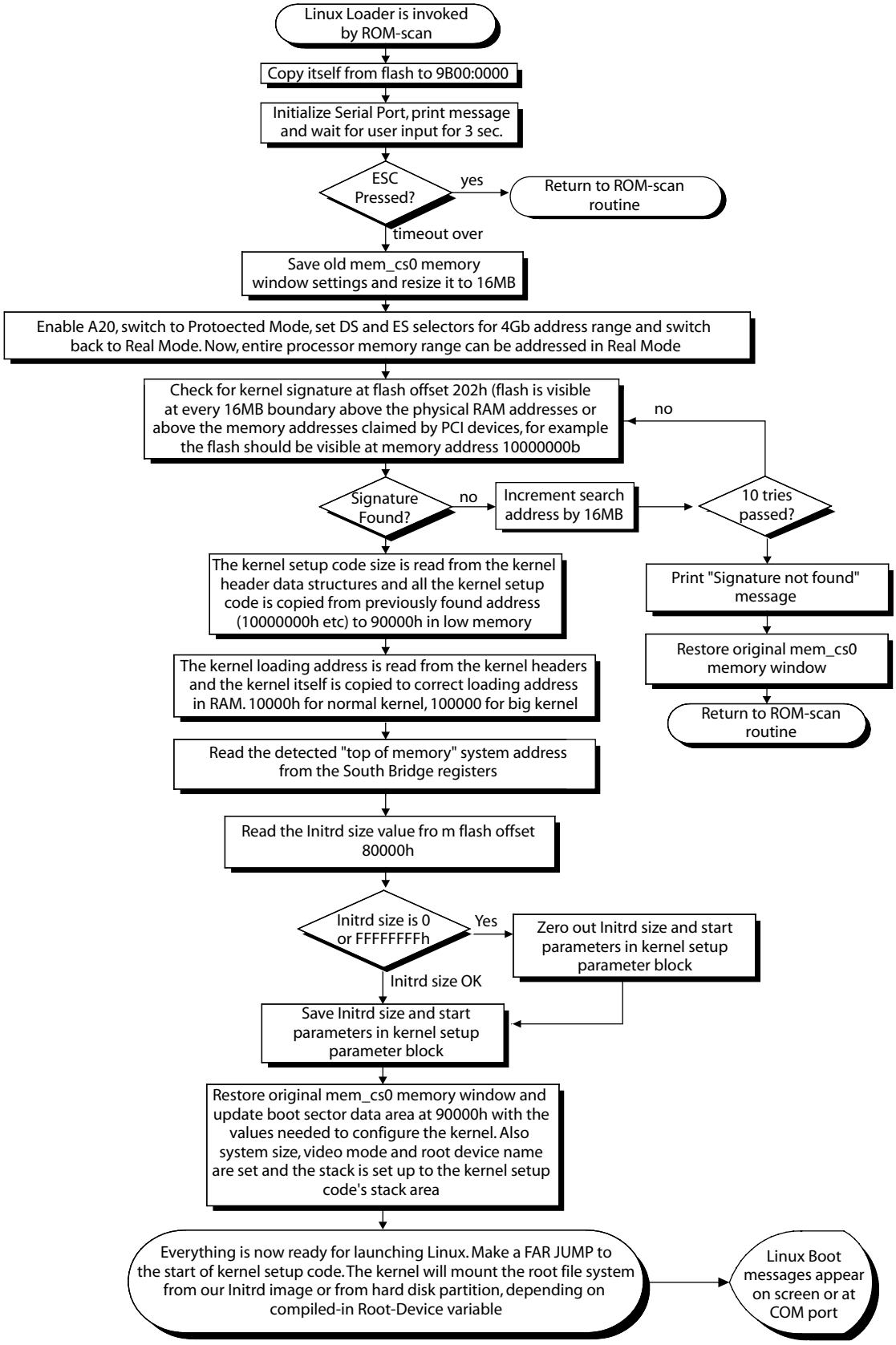
13. The Initrd (RAM-disk image) size and start address are written to the kernel setup parameter block, which resides in memory starting at 90200h. If Initrd is not found, these values will be zeroed out.

14. The previously created 16Mb Flash window is now closed and the original mem_cs0 window restored.

15. In order to boot the kernel normally, the boot sector data area at 90000h is updated with the values needed to configure the kernel. For example, it checks that there are 4 setup sectors, that the root device is read-only, the ram disk is 0, and that the swap size is 0, etc. Also system size, video mode and root device name are set and the stack is set up to the kernel setup code s stack area. The root device name is based on the compiled-in root-device id value, where 100h=ram0, 301h=hda1, 302h=hda2, 303h=hda3, 304h=hda4, and 0=disabled. If the root device id is set to 100h, the root device will be the one contained in compressed form in our Initrd image in Flash. If root device id is set to 301h, then the root device is mounted from the first hard disk partition or /dev/hda1.

16. The last action to start Linux is a Far Jump to the beginning of the kernel setup code. The kernel boot messages will begin appearing on the screen or COM-port and the root file system will be mounted from RAM-disk or the hard disk partition, depending on the compiled-in Root_Device variable. Eventually the user will see a login prompt.

The Linux Loader flowchart



The Flash layout

In order to boot Linux from Flash, we must have multiple items in the flash. The flash layout depends on how the actual hardware is set up and how much Flash memory is available in the system. The items required consist of the following:

- System BIOS or special initialization code (MachZ Phoenix BIOS) which supports Extension-ROMs
- The Linux Loader image, which has been converted to Extension-ROM format with the needed headers and checksums
- The Linux Kernel — exactly the same file created as the end product of the kernel compilation
- Optional Initrd image (a compressed root file system image that will be expanded as a RAM-disk). It can be omitted if the root file system resides on an alternate device (IDE, Compact Flash, Disk on Chip, etc.) and will be mounted from there.

As an example, one possible solution will be to fit all the items to one large Flash-chip such as the 16Mbyte Intel E28F128 StrataFlash. In this case, the Flash memory layout would be as follows:

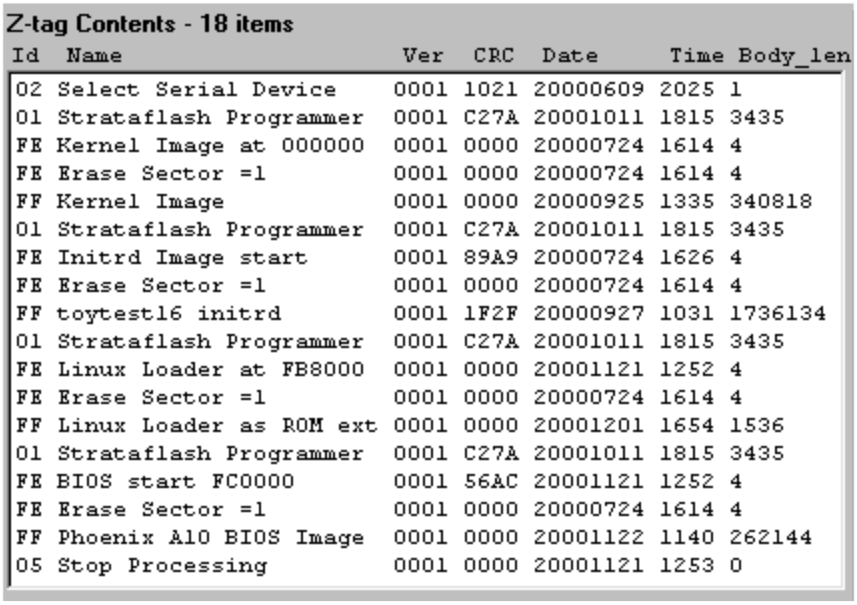
Start offset	Item
FC0000	Phoenix BIOS 256K
FB8000	Linux Loader
080000	Initrd image with 4-byte size at beginning, up to address FB7FFF
000000	Linux Kernel, up to address 07FFFFh

The Linux Loader will be actually mapped at D8000 using the BIOS internal memory chip-select mapping mechanism. The MachZ Phoenix BIOS allows the creation of up to 4 memory windows with chip selects mem_cs0-mem_cs3. Although the BIOS itself uses the mem_cs0 to start from Flash after reset, the BIOS gets shadowed into RAM and this allows us to reprogram mem_cs0 for other memory windows. During the special extension-ROM scan the Linux Loader is mapped to the correct place in RAM and will be found and executed.

In order to use mem_cs0 to map the Linux Loader to the desired place in RAM, use the Phoenix-BIOS Setup Utility ->Advanced -> Advanced chipset Control -> ISA Memory Chip Select Setup and set following values for mem_cs0:

Window Size: 1 - sets window size to 8kb
Window Base: 216 - sets window base to D8000h
Window Page: 3808 — sets Flash page register value to
1000000h—D8000h+FB8000h=(1)EE0000h
omit leading 1 and use only EE0 -> decimal = 3808
Window data width:16 or 8 based upon the data path width of the device used in the design.

Using the Z-Tag Manager Contents window the previously mentioned items can be viewed as follows:



Z-tag Contents - 18 items

Id	Name	Ver	CRC	Date	Time	Body_len
02	Select Serial Device	0001	1021	20000609	2025	1
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	Kernel Image at 000000	0001	0000	20000724	1614	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	Kernel Image	0001	0000	20000925	1335	340818
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	Initrd Image start	0001	89A9	20000724	1626	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	toytest16 initrd	0001	1F2F	20000927	1031	1736134
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	Linux Loader at FB8000	0001	0000	20001121	1252	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	Linux Loader as ROM ext	0001	0000	20001201	1654	1536
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	BIOS start FC0000	0001	56AC	20001121	1252	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	Phoenix A10 BIOS Image	0001	0000	20001122	1140	262144
05	Stop Processing	0001	0000	20001121	1253	0

Figure 1. Z-tag Manager's Z-tag Contents window for defining data to be flashed

This large amount of data can be written to the flash using the Z-Tag Manager s PassThrough mode. Therefore, connect the parallel port extension cable to your development host computer, connect the Z-Tag dongle (set to PassThrough mode) to the extension cable and to your target board s Z-Tag connector. Verify that CS0 is tampered to your selected target flash chip, press the Z-Tag Manager s Write-button and reset the target board to initiate the download and Flash burning sequence. A good practice is to connect the serial cable from the target board s COM1 port to your development host computer s COM port to monitor progress. The COM port settings should be 9600, 8-N-1, handshake none. Use of the Z-Tag Manager is described in detail in the MachZ data book, the Z-Tag Manager manual and other reference documents from ZF.

Conclusion

Starting Linux from Flash is not a complicated task if we have a Linux Loader binary in an Extension-ROM compatible format (there might be several different images for different purposes like mounting root file system from RAM-disk or for mounting the root file system from /dev/hda1).

We need the BIOS or other system initialization code to set-up the hardware properly and detect the amount of memory installed into system. The BIOS or system initialization code will then launch the Linux Loader (or some other operating system loader which can be built using the same general principles) either during the ROM-scan sequence or by directly jumping to it.

In case of a complete Linux system we need to put both the kernel and Initrd images at Flash offsets 0 and 80000h. The Initrd image must have a 4 bytes long image length header before the actual image starts. For mounting the root file system from a hard disk only the kernel is needed in the Flash and Linux Loader has to be compiled with correctly defined root-device id. Generally the current Linux Loader code can be modified to match the HW design and the images can reside in completely different Flash offsets.

Appendix A. The Linux Loader Source Code

```
; BIOS Linux Loader v1.0

       .model tiny
       .486p

; Linux root device options:
; 100h=ram0, 301h=hda1, 302h=hda2, 303h=hda3, 304h=hda4, 0=disabled

Root_Device   equ    100h
Relocate_Loader      equ    1      ; If we have to relocate loader code from flash
                                   ; space before starting
Serial_Addr   equ    03f8h  ; 3F8h = COM1, 2F8h = COM2
Screen_Output equ    1      ; 1 = Output messages also to the screen

SCALL  MACRO  address
       mov    bp,$+6
       jmp    near ptr address
       ENDM

MSG    MACRO  text
       mov    si,offset text
       mov    bp,$+6
       jmp    near ptr Output
       ENDM

PCODE  MACRO  postcode
       mov    al,postcode
       out    80h,al
       ENDM

ZFLWB  MACRO  register,value8
       mov    al,register
       mov    dx,218h
       out    dx,al
       inc    dx
       mov    al,value8
       out    dx,al
       ENDM

ZFLRB  MACRO  register
       mov    al,register
       mov    dx,218h
       out    dx,al
       inc    dx
       in     al,dx
       ENDM

ZFLWDW MACRO  register,value32
       mov    al,register
       mov    dx,218h
```

```
        out     dx,al
        inc     dx
        inc     dx
        mov     eax,value32
        out     dx,eax
        ENDM

ZFLRDW MACRO  register
        mov     al,register
        mov     dx,218h
        out     dx,al
        inc     dx
        inc     dx
        in      eax,dx
        ENDM

        .code
        org     0

Start:

        db      55h,0aah      ; Extension ROM signature,
        db      3             ; and length in 512-byte pages

        PCODE   070h
        mov     ax,cs
        mov     ds,ax
        jmp     Skip_GdtArea

; Global Descriptor Table

        org     10h           ; For proper alignment

Gdt     dd       0,0          ; 1st entry, not used
GdtProtdw     0ffffh,0000h ; 2nd entry
        db      0,93h,8fh,0
GdtDescdw     $-Gdt  ; GDT size
GdtBasedd     0      ; GDT base address

Skip_GdtArea:

IF Relocate_Loader EQ 1

; First lets move our code out of extension ROM space, so we can open new
; 16Mb wide memory window for strataflash where we have kernel and initrd
; images. Since this will overlap memory window for extension ROM where we
; reside at the moment, we need to get out of here.
; Bootsector & Linux kernel setup goes to 9000:0000, length 0A00h bytes,
; Linux kernel itself goes to 1000:0000, max length 07F00h bytes,
; which leaves safe location for us below 1000:0000 or above 9000:0A00,
; so I chose 9B00:0000. We don't have to worry about this if kernel goes into
; high memory (above 1 Mb)
```

```
New_Segequ    9B00h

        mov     ax,New_Seg
        mov     es,ax
        lea     si,Start
        mov     di,si
        mov     cx,offset Loader_End-Start
        cld
        rep     movsb

        db      0eah   ; Far jump to Start
        dw      offset Loader_Start,New_Seg
ENDIF

Loader_Start:

        PCODE  71h

        mov     ax,cs
        mov     ds,ax
        mov     es,ax

; Initialize serial port, so we can communicate

        mov     dx,Serial_Addr+3
        mov     al,80h
        out     dx,al          ; Set DLAB
        mov     dx,Serial_Addr
        mov     ax,12          ; 12 = 9600 bps
        out     dx,ax          ; Baud rate divisor
        mov     dx,Serial_Addr+3
        mov     al,3           ; 3 = 8N1
        out     dx,al          ; Line mode (8N1)
        mov     dx,Serial_Addr+4
        xor     al,al
        out     dx,al          ; Clear DTR & RTS
        MSG     T_Loader_Start; Output loader startup message

; Here we wait 3 seconds for user input, if ESC key is pressed, loader quits
; with jump to the original Int 19h vector

        mov     ax,40h
        mov     es,ax
        mov     ebx,es:[ 6ch]
        add     ebx,36         ; We wait 55 timer ticks, ca 3 seconds
@@:
        cmp     ebx,es:[ 6ch]
        jl      @f
        mov     ah,1
        int     16h            ; Check if there is anything in keyboard buffer
        jz      @b
        mov     ah,0           ; If yes, fetch it and compare to ESC keycode
        int     16h
```

```
        cmp     al,27           ; If it was ESC, quit loader
        jne     @f
        MSG     T_Cancel

; Exit loader and continue with normal boot sequence

        PCODE   72h
        retf
@@:
        MSG     T_Start

        PCODE   73h

; Save current memory window settings

        lea     edi,offset MemWinISA24
        ZFLRB   5Bh             ; ISA 24-bit address calculation
        stosb
        ZFLRDW  26h             ; Window base
        stosd
        ZFLRDW  2Ah             ; Window size
        stosd
        ZFLRDW  2Eh             ; Window page
        stosd

; Define 16Mb wide memory window for chip select 0

        ZFLWB   5Bh,1           ; Set ISA 24-bit address calculation
        ZFLWDW  26h,0           ; Set base address (actual ports 27h and 28h)
        ZFLWDW  2Ah,1000000h-1; Window size is 16MB (strataflash)
        ZFLWDW  2Eh,0           ; Page address

; Enable A20 line

        PCODE   74h

         cli
        in      al,92h
         jmp    $+2
         jmp    $+2
        or      al,00000010b ; Enable A20 bit
        out     92h,al

; Initialize and load GDT

        PCODE   75h

        xor     eax,eax
        mov     ax,cs
        shl     eax,4
        add     eax,offset Gdt
        mov     cs:GdtBase,eax
        lgdt    fword ptr cs:GdtDesc
```

```
; Switch processor to protected mode

        mov     eax,cr0
        mov     ebx,eax
        or      ax,1            ; Set PE bit
        mov     cr0,eax         ; Enable protected mode
        jmp     $+2             ; Flush instruction cache
        mov     ax,(GdtProt - Gdt)
        mov     ds,ax           ; Define selectors for DS & ES
        mov     es,ax

; Switch processor back to real mode

        mov     cr0,ebx         ; Clear PE bit, back to real mode
        jmp     $+2             ; Flush instruction cache

; Check for Linux kernel setup signature. If we cant find the signature in
; first try, we perform a scan loop on higher addresses just to be sure that
; the address space where we were looking was not claimed by any other device
; with higher priority. This scanning technique is possible because of ISA bus
; being only 24 bits wide and its 16Mb address space gets repeated after every
; 16Mb block through entire 4GB adress space. We start from 10000000h, thats
; above 256Mb, maximum amount of RAM that MachZ can be configured with.

        PCODE   76h

        mov     esi,10000000h; Start address
        mov     cx,10           ; Number of cycles
@@:
        add     esi,202h        ; Start address + kernel setup signature offset
        mov     eax,[esi]
        cmp     eax,053726448h; Look for 'HdrS'
        je      SigFound
        add     esi,01000000h; Add 16Mb to the address and try again
        loop    @b
        jmp     NoSignature     ; No signature found, skip the whole thing

; Now copy kernel setup and bootstrap code

SigFound:

        sub     esi,202h
        mov     cs:Kernel_Start,esi ; Kernel setup start address

        PCODE   77h

        add     si,1f1h
        xor     ax,ax
        mov     al,[esi]        ; Get setup sector size
        inc     al              ; Add bootstrap sector
        shl     ax,9            ; Multiply by 512 for size in bytes
        mov     bx,ax           ; Store value for kernel start address
```

```
        sub     esi,1f1h        ; Start address of the kernel setup code
        mov     edi,90000h      ; Destination address
        mov     cx,ax
@@:
        mov     eax,ds:[esi]
        mov     ds:[edi],eax
        add     esi,4
        add     edi,4
        sub     cx,4
        jnz     @b

; Now copy kernel image

        PCODE  78h

        mov     esi,cs:Kernel_Start
        add     esi,211h
        mov     al,[esi]        ; Kernel boot option
        add     esi,3
        mov     edi,[esi]       ; Kernel load offset in system memory
        sub     esi,214h
        mov     si,bx           ; Start address of the kernel image
        mov     ecx,080000h     ; Maximum kernel size to copy
        or      al,al
        jnz     @f
        shl     edi,4           ; Start address of kernel (10000h or 100000h)
@@:
        mov     eax,[esi]
        mov     [edi],eax
        add     esi,4
        add     edi,4
        sub     ecx,4
        jnz     @b

        PCODE  79h

; Read top of system memory address from south bridge
; This is specific to the MachZ BIOS'es

        mov     eax,8000904Ch; PCI south-bridge top of system memory register
        mov     dx,0CF8h
        out     dx,eax
        mov     dx,0CFCh
        in      eax,dx
        and     al,0f0h         ; We have to clear lower 4 bits (SB speciality)
        dec     eax

; Check for initrd size/presence in flash rom,
; and copy it to the top of system memory

        PCODE  7Ah

        mov     edi,eax
```

```
        mov    esi,cs:Kernel_Start
        add    esi,80000h    ; Start address of the initrd image in memory window
        mov    ecx,[esi]     ; Get size of the initrd image
        cmp    ecx,0         ; Skip initrd if size is zero, means it's disabled
        jz     SkipInitrd
        cmp    ecx,0FFFFFFFFh; Also skip initrd if the memory is pobably
        jz     SkipInitrd    ; not initialized

        PCODE  7Bh

        add    esi,4         ; Skip first 4 bytes of image (initrd size)
        neg    ecx
        add    edi,ecx       ; Calculate start address of the image in system memory
        neg    ecx
        xor    di,di
        mov    ebx,edi       ; Save start address of the image
@@:
        mov    eax,[esi]
        mov    [edi],eax
        add    esi,4
        add    edi,4
        sub    ecx,4
        jc     @f
        jnz    @b
@@:
        mov    esi,cs:Kernel_Start
        add    esi,80000h    ; Start address of the initrd image in memory window
        mov    ecx,[esi]     ; Get size of the initrd image
        jmp    @f

SkipInitrd:

        PCODE  7Ch

        sub    ebx,ebx       ; Set zeroes if initrd was not found in flash
        sub    ecx,ecx

; Write start address and size of ramdisk image (initrd) to the kernel setup
; parameter block

@@:
        PCODE  7Dh

        mov    ax,9020h      ; Kernel setup segment
        mov    ds,ax
        mov    ds:[24],ebx   ; Start address
        mov    ds:[28],ecx   ; Image size

; Restore original memory window

        ZFLWB  5Bh,0         ; Clear full 24-bit ISA addressing
        ZFLWDW 2Eh,0F00000h  ; Set page
        ZFLWDW 2Ah,10000h-1  ; Window size is 64k
```

```
        ZFLWDW 26h,0F0000h   ; Set base address

; Setup parameters

        mov    ax,9000h      ; Bootsector data area
        mov    ds,ax
        mov    byte ptr ds:[ 1f1h],4       ; Setup sectors
        mov    word ptr ds:[ 1f2h],1       ; Root flags (read only)
        mov    word ptr ds:[ 1f4h],7f00h   ; System size
        mov    word ptr ds:[ 1f6h],0       ; Swap device
        mov    word ptr ds:[ 1f8h],0       ; Ramdisk
        mov    word ptr ds:[ 1fah],0f00h   ; VGA screen mode
        mov    word ptr ds:[ 1fch],Root_Device    ; Root file system device

; Command line patch

        mov    ds:[ 020h],0a33fh
        mov    ds:[ 022h],8cc1h

; Root device name

        cld
        mov    si,offset T_Root_Device
        mov    di,08cc1h
        mov    ax,ds
        mov    es,ax         ; ES=9000h
        mov    ax,cs
        mov    ds,ax         ; DS=CS
        mov    cx,16
        rep    movsb

        mov    ax,9020h      ; Kernel setup segment
        mov    ds,ax
        mov    ss,ax         ; Put stack at 09020h:0x4000-12.
        mov    sp,4000h-12   ; 0x4000 is arbitrary value >= length of
                             ; bootsect + length of setup + room for stack
                             ; 12 is disk parm size

        mov    byte ptr ds:[ 16],61h; Set loader type and version

; Everything is done, now lets jump into kernel setup code

        PCODE  7Eh

        db     0eah          ; Far jump into kernel setup code
        dw     0,09020h

NoSignature:

        PCODE  7Fh
        MSG    T_SigNotFound

; Restore original memory window
```

```
        lea     si,offset MemWinISA24
        lodsb
        ZFLWB   5Bh,al          ; Clear full ISA addressing bit
        lodsd
        ZFLWDW  2Eh,eax         ; Window page
        lodsd
        ZFLWDW  2Ah,eax         ; Window size
        lodsd
        ZFLWDW  26h,eax         ; Window base
        retf

;-------------------------------------------------------------
;
; Output string from CS:SI ending with zero
;

Output:
        mov     dx,Serial_Addr+5
@@:
        in      al,dx
        test    al,20h
        jz      @b
        mov     al,byte ptr cs:[ si]
        inc     si
        cmp     al,0
        jne     @f
        jmp     bp
@@:
        sub     dx,5
        out     dx,al
IF Screen_Output EQ 1
        mov     ah,0eh
        int     10h
ENDIF
        jmp     short Output

MemWinISA24   db      0
MemWinBase    dd      0
MemWinSize    dd      0
MemWinPage    dd      0

Kernel_Start  dd      0

IF Root_Device EQ 100h
T_Root_Device db      '/dev/ram0 ',0,0,0,0,0,0
ENDIF
IF Root_Device EQ 301h
T_Root_Device db      '/dev/hda1 ',0,0,0,0,0,0
ENDIF
IF Root_Device EQ 302h
T_Root_Device db      '/dev/hda2 ',0,0,0,0,0,0
ENDIF
```

```
IF Root_Device EQ 303h
T_Root_Device db     '/dev/hda3 ',0,0,0,0,0,0
ENDIF
IF Root_Device EQ 304h
T_Root_Device db     '/dev/hda4 ',0,0,0,0,0,0
ENDIF
IF Root_Device EQ 0
T_Root_Device dd     0,0,0,0
ENDIF

T_Loader_Startdb     13,10,'BIOS Linux loader, press ESC to cancel...',0
T_Start       db     'starting.',13,10,10,0
T_Cancel      db     'cancel.',13,10,10,0
T_SigNotFound db     'Linux kernel setup signature not found.',13,10,10,0

Loader_End:
              end    Start
```